Assignment 1 CS170. Introduction to Artificial Intelligence

Christopher Hyun Instructor: CS Eamonn Keogh

ID 861162836

chyun001@ucr.edu

02-November-2018

In completing this assignment, I consulted…

- PowerPoints' Blind Search.ppt, Heuristic Search.ppt, and Search Review_for_A-Start.ppt to get a better understanding of the search algorithms (especially A*).
- Watched this video (https://www.youtube.com/watch?v=6TsL96NAZCo) to further reinforce my understanding of A*.
- Looked at the example report to help format my report and used an example puzzle on page 5 (puzzle 3) to test my own code.

All the important code is original. Unimportant subroutines that are not completely original are…

- The C++ library "iostream" (used "cout" and "cin").
- The C++ library "vector" (used to store the puzzle).

Report

Overview

The eight-puzzle game is a three by three board game where eight numbers (one to eight) occupy a space on the 3 x 3 board where there is one blank space. The goal of the board game is to slide around the pieces until they are numbered in ascending order from top to bottom. For this project, I have created a program in C++ that solves the eight-puzzle using the following algorithms:

- Uniform cost search (UCS)
- A* Search (Misplaced Tile Heuristic)
- A* Search (Manhattan Distance Heuristic)

Below, I will discuss what I have discovered using these algorithms and the optimality between them. Before that, here is a brief description on each algorithm.

Uniform Cost Search

Uniform cost search is a search algorithm where it expands the cheapest node. The cost of each node is designated in g(n). In our eight-puzzle case, each node costs exactly one to expand.

A* Search (Misplaced Tile)

A* search is similar to that of uniform cost search in the sense that it expands the cheapest node. However, the difference is that it takes in a heuristic to help predict how far away you are from the goal. One way to calculate the heuristic is to count all the misplaced tiles (excluding the blank spot) on the puzzle board. After evaluating the heuristic, which is denoted by h(n), you add it to the cost g(n) and expand the cheapest node. Note, hard coding h(n) to zero degenerates the algorithm to uniform cost search.

Example:

```
[ [1 0 2],
  [4 5 3],
  [7 8 6] ]       h(n) = 3
```

A* Search (Manhattan Distance)

Same algorithm as the one above except that the heuristic is evaluated differently. Another way to calculate the heuristic is to use something called the "Manhattan Distance." To calculate this, for each misplaced tile, move each number tile to the correct spot as if there are no numbers blocking the way. For each move, increment the heuristic (h(n)) by one.

Example:

```
[ [0 1 2],
  [4 5 3],
  [7 8 6] ]      h(n) = 4
```

Comparison

Now, let's compare the algorithms and see how they hold up against each other. I will be categorizing them in four different categories

1. The number of nodes expanded.
2. The maximum number of nodes in the vector at any one time.
3. The depth where the solution was found. (In our case, the depth also doubles as the cost)
4. The time it took to solve the problem.

Test Number #1

```
[ [0 1 2],
  [4 5 3],
  [7 8 6] ]
```

	UCS	A* Misplaced	A* Manhattan
Expanded	29	4	4
Max Stored	18	4	4
Depth/Cost	4	4	4
Time	< 1 Second	< 1 Second	< 1 Second

Although uniform cost search expanded seven times more than A* Misplaced and A* Manhattan, all three solved the puzzle very fast and there was no notable difference in terms of time. Let's try a bit harder example.

Test Number #2

[[3 5 8],

[4 2 6],

[0 1 7]]

	UCS	A* Misplaced	A* Manhattan
Expanded	X	3178	374
Max Stored	X	2013	241
Depth/Cost	X	20	20
Time	> 10 Minutes Terminated	Approximately 2 Minutes	Approximately 1 Second

After this test, we can see that there is a big difference in runtime here. Uniform cost search could not find a solution within ten minutes of running. Both A* algorithm's on the other hand did find a solution but the heuristic evaluation between them made all the difference in terms of speed. Now let's try an even harder puzzle.

Test Number #3 (oh boy)

[[8 7 1],

[6 0 2],

[5 4 3]]

	UCS	A* Misplaced	A* Manhattan
Expanded	X	X	946
Max Stored	X	X	614
Depth/Cost	X	X	22
Time	> 10 Minutes Terminated	> 10 Minutes Terminated	Approximately 4 seconds

As we can see here, both uniform cost search and A* using the Misplaced Tile Heuristic falls off dramatically compared to A* using the Manhattan Distance. While A* Manhattan is able to provide a solution in four seconds, the other two could not find one within 10 minutes.

Conclusion

Going off the test results from above, we can see that fastest algorithm is the A* search using the “Manhattan Distance” heuristic. Although the same algorithm was used, A* search using the “Misplaced Tile” heuristic expanded more nodes which resulted in a slower time. This means that that the quality of the heuristic matters when finding a solution. Uniform cost search, which is just A* with $h(n) = 0$, was worse than A* using either heuristic evaluation method.

Manhattan Distance > Misplaced Tile > Uniform Cost Search